Self-Consistent Langevin Simulation of Coulomb Collisions in Charged-Particle Beams

Ji Qiang*, Robert D. Ryne*, and Salman Habib†
*LANSCE-1, LANSCE Division, MS H817, Los Alamos National Laboratory, Los Alamos, New Mexico 87545
† T-8, Theoretical Division, MS B285, Los Alamos National Laboratory, Los Alamos, New Mexico 87545
(July 26, 2000)

In many plasma physics and charged-particle beam dynamics problems, Coulomb collisions are modeled by a Fokker-Planck equation. In order to incorporate these collisions, we present a three-dimensional parallel Langevin simulation method using a Particle-In-Cell (PIC) approach implemented on high-performance parallel computers. We perform, for the first time, a fully self-consistent simulation, in which the friction and diffusion coefficients are computed from first principles. We employ a two-dimensional domain decomposition approach within a message passing programming paradigm along with dynamic load balancing. Object oriented programming is used to encapsulate details of the communication syntax as well as to enhance reusability and extensibility. Performance tests on the SGI Origin 2000, IBM SP RS/6000 and the Cray T3E-900 have demonstrated good scalability. A test example for an initially anisotropic beam approaching thermal equilibrium is given.

PACS numbers: 52.75

I. INTRODUCTION

Coulomb collisions play an important role in many areas of plasma physics, accelerator physics, and astrophysics. The long-range nature of the force leads to a fundamental difference between how such collisions need to be treated compared to the Boltzmann approach familiar when dealing with dilute neutral gases. Since most collisions occur at large impact parameters, the particle deflection per collision is small. Moreover, at any given time, a particular particle is interacting with many other particles. For these reasons, a simple Boltzmann picture of the collisions is not applicable (the Boltzmann collision integral diverges at large distances).

When 'soft' collisions such as those described above are encountered, the appropriate transport equation is of the Fokker-Planck form [1]. For the case of Coulomb collisions between charged particles, the derivation of the appropriate Fokker-Planck equation is somewhat delicate. Depending on one's taste and notions of rigor, several different methods may be employed: the fundamental Boltzmann kernel may be expanded in powers of momentum transfer and effectively linearized [2]; the BBGKY formalism may be utilized with an expansion in powers of the Coulomb logarithm used to truncate the expansion at second order [3]; and a simple master equation-like argument may also be used to derive the Fokker-Planck collision kernel [1]. Fortunately, all these derivations lead to essentially the same final result.

In many cases of physical interest, such as intense beams, one needs to take into account the mean force field of all other particles on the particle of interest (the Vlasov-Poisson equation) as well as account for the soft collisions. The inclusion of a Fokker-Planck collision term on the right hand side of the Vlasov equation gives rise to the Landau equation. The Landau equation is a partial differential equation with self-consistently determined systematic force terms as well as external fields, if present, and self-consistent friction and diffusion coefficients arising from the Fokker-Planck treatment of collisions. Determination of all the self-consistent contributions requires the computation of convolution integrals in either real or velocity space.

A successful approach to modeling the Vlasov-Poisson equation is the popular PIC technique where simulation particles are used to indirectly represent the phase space distribution function and the Poisson equation is solved on a spatial grid. The advantages of the PIC method include its relative conceptual simplicity, high performance resulting from fast Poisson solvers, relatively low memory cost for the grid ($O(L^k)$ where k is the number of spatial dimensions), and insensitivity to the generation of small-scale structure in the distribution function. Moreover, PIC simulations for accelerator applications have been implemented efficiently on parallel computing platforms [4]. Fokker-Planck collisions can be included in the PIC method via the addition of friction and (multiplicative) stochastic forces in the equations of motion for the simulation particles: This is the Langevin approach to incorporating soft collisions. It should be kept in mind that numerical collisions are present in any PIC simulation of the type just described. Thus, it is appropriate to include the physical collisions only when the numerical collisions are strongly suppressed in the original Vlasov-Poisson simulation. This condition can be met in some situations of interest [5].

The main difficulty in carrying out the Langevin PIC program is the fact that the self-consistent friction and diffusion coefficients themselves depend on the velocity, thus, in principle, for every simulation particle one

needs to carry out two convolution integrals in velocity space followed by appropriate derivatives, also in velocity space. Given the PIC point of view, one would wish to introduce a velocity grid associated with each spatial grid cell, carry out the convolutions on the velocity grid and then use interpolation to determine the appropriate friction and diffusion coefficients for the simulation particles belonging to that particular spatial grid cell. These tasks have been viewed as being much too difficult to actually carry out: either the Spitzer approximation has been employed [6] or an isotropic velocity distribution has been assumed for the scattering particles [7].

However, on modern parallel machines these problems can be overcome (in large part) and the fully self-consistent friction and diffusion coefficients obtained numerically for any distribution. The purpose of the present paper is to explain and demonstrate how this can be achieved. In short, the key points are that the velocity grids need not be very large (we found 32^3 to be sufficient), one may associate a single velocity grid not with a single spatial grid cell but with some number of them (a form of coarse-graining), the number of particles associated with each spatial 'super-cell' is large enough to guarantee low sampling noise in velocity space, and finally, the convolution and interpolation strategies already implemented for the spatial part of the Vlasov-Poisson equation may be directly extended to velocity space.

This paper is organized as follows. The Landau equation and the numerical methods are presented in Section 2. The parallel implementation is outlined in Section 3, performance tests given in Section 4, and results reported in Section 5.

II. THE LANDAU EQUATION AND ITS NUMERICAL SOLUTION

The Landau equation for the evolution of the single-particle distribution function f is of the form [3]:

$$\frac{\partial f}{\partial t} + \mathbf{v}\cdot\frac{\partial f}{\partial \mathbf{r}} + \frac{\mathbf{F}}{m}\cdot\frac{\partial f}{\partial \mathbf{v}} = -\frac{\partial}{\partial \mathbf{v}}\cdot\mathbf{F_d} f + \frac{1}{2}\frac{\partial^2}{\partial \mathbf{v}\partial \mathbf{v}} : \mathbf{D} f \tag{1}$$

where t is the time, $\mathbf{r}$ is the spatial vector, $\mathbf{v}$ is the velocity vector, m is the mass of particle, $\mathbf{F_d}$ is the dynamic friction coefficient and $\mathbf{D}$ is the diffusion coefficient. They are defined via

$$\mathbf{F_d} = \frac{ne^4}{4\pi\epsilon_0^2 m^2}\lambda\frac{\partial H}{\partial \mathbf{v}} \tag{2}$$

$$\mathbf{D} = \frac{ne^4}{4\pi\epsilon_0^2 m^2}\lambda\frac{\partial G}{\partial \mathbf{v}\partial \mathbf{v}} \tag{3}$$

$$\lambda = ln(\frac{mv^2\lambda_D}{2e^2}) \tag{4}$$

$$H = 2\int \mathbf{d}^3\tilde{\mathbf{v}}\frac{f(\mathbf{r},\tilde{\mathbf{v}})}{|\mathbf{v}-\tilde{\mathbf{v}}|} \tag{5}$$

$$G = \int \mathbf{d}^3\tilde{\mathbf{v}} f(\mathbf{r},\tilde{\mathbf{v}})|\mathbf{v}-\tilde{\mathbf{v}}| \tag{6}$$

with n being the particle density, λ being the Coulomb logarithm, and $\lambda_D = \sqrt{kT/\epsilon_0 ne}$, the Debye length. Here, k is the Boltzmann constant, T is the temperature, ϵ_0 is the vacuum dielectricity, e is the charge of particle. The force $\mathbf{F}$ includes both the external force and the self-generated mean field space charge force which can be obtained from the Poisson equation:

$$\nabla^2\phi(\mathbf{r}) = -\frac{\rho(\mathbf{r})}{\epsilon_0} \tag{7}$$

and

$$\rho(\mathbf{r}) = \int \mathbf{d}^3\mathbf{v} f(\mathbf{r},\mathbf{v}) \tag{8}$$

Here, ϕ is the electric potential and ρ is the charge density.

The stochastic (multiplicative noise) particle equations of motion that follow from the Landau equation are (Cf. Ref. [8])

$$\mathbf{r}' = \mathbf{v}, \tag{9}$$

$$\mathbf{v}' = \frac{\mathbf{F}}{m} + \mathbf{F_d} + \mathbf{Q}\cdot\mathbf{\Gamma}(t), \tag{10}$$

where $\mathbf{\Gamma}(t)$ are Gaussian random variables with

$$\langle\Gamma_i(t)\rangle = 0, \tag{11}$$

$$\langle\Gamma_i(t)\Gamma_j(t')\rangle = \delta_{ij}\delta(t-t'). \tag{12}$$

The matrix $\mathbf{Q}$ is related to the diffusion coefficient $\mathbf{D}$ by $D_{ij} = Q_{ik}Q_{jk}$. The Q_{ik} can be obtained using an orthogonal transformation, taking the positive root of the eigenvalues and then transforming back.

The friction and diffusion coefficients follow from Eqns. (2) - (6). Computation of H and G requires carrying out convolution integrals. To do this we employ a PIC charge deposition onto a velocity grid using a linear scheme to get the distribution function f on the grid. This is followed by a FFT-based convolution which requires doubling the computational grid in each velocity direction in order to correctly impose open boundary conditions [9]. The friction and diffusion coefficients can now be computed on the grid using second-order central finite differences. These coefficients are then reinterpolated back onto the particles using the original linear PIC scheme. The self-generated space charge forces are also calculated by depositing particles onto a spatial grid following the PIC approach. The scalar potential in the Poisson equation is solved following the same FFT-based method explained above and the force on the particles obtained by numerical differentiation an reinterpolation.

This force together with the external force field and the forces due to dynamic friction force and diffusion are used to advance the charged particles for one time-step using a (stochastic) leap-frog algorithm.

III. OBJECT-ORIENTED PARALLEL IMPLEMENTATION

We employ a two-dimensional domain decomposition approach following Refs. [10]. A schematic plot of the two-dimensional decomposition on the y-z plane is shown in Fig. 1. The solid grid lines define the computational domain grids. The dashed lines define the local computational domain on each processor. The boundary grids are the outer-most grids inside the physical boundary. Guard grids are used for temporary storage of grid quantities from neighboring processors. The physical computational domain is defined as a 3-dimensional rectangular box with range $x_{min} \le x \le x_{max}$, $y_{min} \le y \le y_{max}$, and $z_{min} \le z \le z_{max}$. This domain is decomposed on the $y-z$ plane into a number of small rectangular blocks and these blocks are mapped to a logical two-dimensional Cartesian processor grid, one rectangular block per processor. The range of a block on a single processor is defined as $x_{min} \le x \le x_{max}$, $y_{lcmin} \le y \le y_{lcmax}$, and $z_{lcmin} \le z \le z_{lcmax}$. The subscripts $lcmin$ and $lcmax$ specify local minima and maxima. The mesh grid stores field-related quantities such as charge density and electric field. The number of grid points along three dimensions on a single processor is defined as:

$$Nx_{local} = int[(x_{max} - x_{min})/h_x] + 1 \tag{13}$$

$$Ny_{local} = int[(y_{lcmax} - y_{min})/h_y] - int[(y_{lcmin} - y_{min})/h_y] + N_g \tag{14}$$

$$Nz_{local} = int[(z_{lcmax} - z_{min})/h_z] - int[(z_{lcmin} - z_{min})/h_z] + N_g \tag{15}$$

where h_x, h_y, and h_z are the mesh sizes along the x, y and z directions, respectively. The quantity N_g refers to the number of guard grids in Ny_{local} and Nz_{local}. $N_g = 2$ if the number of processors in that dimension is greater than 1; otherwise, $N_g = 1$. Particles with spatial positions within the local computational boundary are assigned to the processor containing that part of the physical domain.

The parallel computation starts with constructing a 2-D logical Cartesian processor grid, reading input data from processor 0 and broadcasting it to the other processors, setting up the local initial computational domain, initializing objects, and generating particles from the initial distribution function. The particles generated on each processor advance following each time step. If a particle moves outside the local computational domain, it is sent to the corresponding processor where it is now located. A particle manager function handles explicit communication using MPI. The y and z positions of every particle on each processor are checked. The particle is copied to one of its four buffers and sent to one of its four neighboring processors when its y or z position is outside the local computational domain. After a processor receives the particles from its neighboring processors, it decides among those particles whether some of them will be further sent out or not. The outgoing particles are counted and copied into four temporary arrays. The remaining particles are copied into another temporary array. This process is repeated until there is no outgoing particle found on all processors. Finally, the particles in the temporary storage along with the particles left in the original particle array are copied into a new particle array.

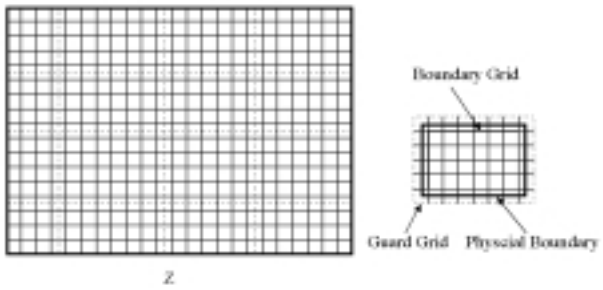

FIG. 1. Schematic of the the 2-D domain decomposition in the $y-z$ domain.

After each particle moves to its local computational domain, a linear particle-deposition scheme is carried out for all processors to obtain the charge density on the grid. Particles located between the boundary grid and computational domain boundary will also contribute to the charge density on the boundary grids of neighboring processors. Hence, explicit communication is required to send the charge density on the guard grids, which is from the local particle deposition, to the boundary grids of neighboring processors to sum up the total charge density on the boundary grids. With the charge density on the grids, Hockney's FFT algorithm [9] is used to solve the Poisson equation with open boundary conditions. This algorithm requires the original grid number to be doubled in each dimension. The charge density on the original grid is kept the same, and the charge density elsewhere is set to 0. The Green's function on the original grid is defined as

$$G_{p,q,r} = \frac{1}{\sqrt{(h_x(p-1))^2 + (h_y(q-1))^2 + (h_z(r-1))^2}}, \tag{16}$$

where $p = 1, \ldots, Nx+1$, $q = 1, \ldots, Ny+1$, $r = 1, \ldots, Nz+1$. Here, Nx, Ny, Nz are the total computation grid numbers without including guard grids in

all three dimensions. For points outside the original grid, symmetry is used to define the Green's function according to

$$G_{p,q,r} = G_{2Nx-p+2,q,r}, \tag{17}$$
$$G_{p,q,r} = G_{p,2Ny-q+2,r}, \tag{18}$$
$$G_{p,q,r} = G_{p,q,2Nz-r+2}, \tag{19}$$

where $p = Nx+2,\ldots,2Nx$, $q = Ny+2,\ldots,2Ny$, $r = Nz+2,\ldots,2Nz$. Communication is required to double the original distributed 3-dimensional grid explicitly. This can be avoided by including this process into the 3-dimensional FFT. In the 3-dimensional parallel FFT, we have taken advantage of the undistributed dimension along the x dimension, where a local serial FFT can be done in that dimension for all processors. A local temporary two-dimensional array with size $(2Nx, Ny_{local})$ is defined to contain part of the charge density at fixed z. The charge density on the original grid is copied into the (Nx, Ny_{local}) part of the temporary array. The rest of the temporary array is filled with 0. In regard to the FFT of the Green's function, symmetry can again be used to obtain the values of the Green's function in the region $(Nx+2, Ny_{local})$. After the local two-dimensional FFT along x is done, it is copied back to a slice of a new 3-dimensional array with size $(2Nx, Ny_{local}, Nz_{local})$. A loop through Nz_{local} gives the FFT along x for the three dimensional array. This is followed by a transpose to switch the x and y indices. Now, the 3-dimensional matrix has size $(Ny, Nx'_{local}, Nz_{local})$ where Nx'_{local} is the new local number of grids in the x dimension along the y dimension processors. A similar procedure yields the FFT along the y direction for a doubled grid of size $(2Ny, Nx'_{local}, Nz_{local})$. Another transpose is used to switch the y and z indices and a local FFT along z with a double-size grid is done on all processors to finish the 3-dimensional FFT for the double-size grid in all three dimensions. During the inverse parallel FFT, a reverse process is employed to obtain the potential on the original grids.

From the potential on the grid, we calculate the electric field using central finite differences. To calculate the electric field on a boundary grid, the potential on a boundary grid of neighboring processors is required. A communication pattern similar to that employed in the charge density summation on the boundary grids is used to send the potential from the boundary grids to the guard grids of neighboring processors. After the electric field on the grids is obtained, the local particle-push requires interpolation from the grids onto the local particles. Since we have used a linear PIC scheme, the electric field of particles between the boundary grid and computational domain boundary will also depend on the electric field on the boundary grid of neighboring processors. A similar communication pattern is used to send the electric field from the boundary grids to the guard grids of the neighboring processors. With the electric field on grids local to each processor, interpolation is done for all processors to obtain the space-charge force on every particle. The dynamic friction coefficient and diffusion coefficient are calculated on each processor. The local computational grids are lumped into a small number of subdomains (the super-cells). Particles within each subdomain will have the same friction and diffusion coefficients. A three-dimensional velocity grid is built on each subdomain for all particles in this domain. Following the scheme described in Section 2, we compute the friction and diffusion coefficients on all processors and reinterpolate them back to the local particles. The local particles are then updated in velocity space.

Dynamic load balancing is employed with adjustable frequency to keep the number of particles on each processor approximately equal. A density function is defined to find the local computational domain boundary so that the number of particles on each processor is roughly balanced. This number depends on the local integration of the charge density on each processor. To determine the local boundary, first, the three-dimensional charge density is summed up along the x direction on each processor to obtain a two-dimensional density function. This function is distributed locally among all processors. Then, the two-dimensional density function is summed up along the y direction to get the local one-dimensional charge density function along z. This density function is broadcast to the processors along the y direction. The local charge density function is gathered along z and broadcast to processors along the z direction to get a global z direction charge density distribution function on each processor. Using this global z direction density distribution, the local computational boundary in the z dimension can be determined assuming that each processor contains a fraction of the total number of particles, about equal to $1/nproc_z$, where $nproc_z$ is the number of processors along the z direction in the two-dimensional Cartesian processor grid. A similar process is used to determine the local computational boundary in the y direction. Strictly speaking, the above algorithm will work correctly for a two-dimensional density distribution function which can be separated as a product of two one-dimensional functions along each direction. However, our experience has been that this algorithm works reasonably well for a broad range of distributions.

The simulation implemented uses object-oriented programming in C++. Based on our previous experience of object-oriented software design for linear accelerator beam dynamics simulations, we have defined a particle manager class, *Ptclmger.C* to move particles among the processors, a field data exchanger class, *Fldexch.C*, to communicate the neighboring data, a utility class, *Utility*, to manage global communication in the matrix transpose, a input-output handler class, *InOut.C* to interface with the outside environment, and a two-dimensional

Cartesian processor class, *Pgrid2d.C*. These classes work together as a low level class to encapsulate communication details used in the parallel message passing programming paradigm. High level application classes, the beam class, field class and beam line element class, are built on top of the low level classes without knowing the details of the communication. Polymorphism is used to access concrete beam line elements, e.g. quadrupole, in the beam line element class definition. A simulation manager class, *AccSimulator.C*, is defined to run the simulation.

IV. PERFORMANCE TESTING

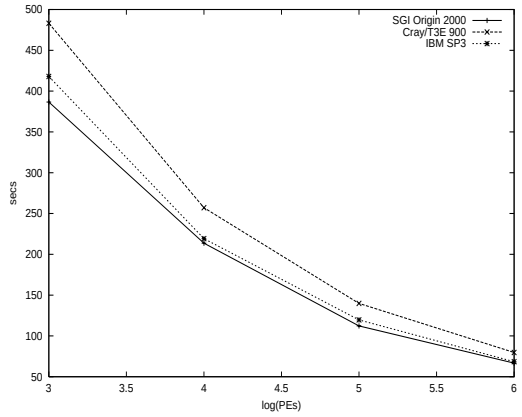

FIG. 2. Time cost as a function of number of processors on the Origin 2000 and the T3E-900.

The parallel performance of the simulation code was tested on a distributed memory machine, the Cray T3E-900, and on a distributed shared memory machine, the SGI Origin 2000 and IBM SP RS/6000. Fig. 2 gives the time cost as a function of number of processors on these machines. The total numerical particle number is two million with a $64 \times 64 \times 64$ spatial grid for the electric field solver, 8^2 super-cells and a $32 \times 32 \times 32$ velocity grid for the dynamic friction and diffusion coefficients. Good scalability is obtained on both machines. The slightly better performance on the SGI Origin may be due to the much larger secondary cache (4 MB) than that of Cray T3E (100 KB) and faster clock speed (250 MHz) than that of IBM SP (200 MHz). To investigate the effect of problem size on the scalability, we tested the code with an increased spatial grid 8^3 for the dynamic friction and diffusion coefficients. Fig. 3 gives the speedup (normalized by the time on eight processors) on the SGI Origin 2000 as a function of number of processors for two different problem sizes. Increasing the problem size improves the scalability of the code.

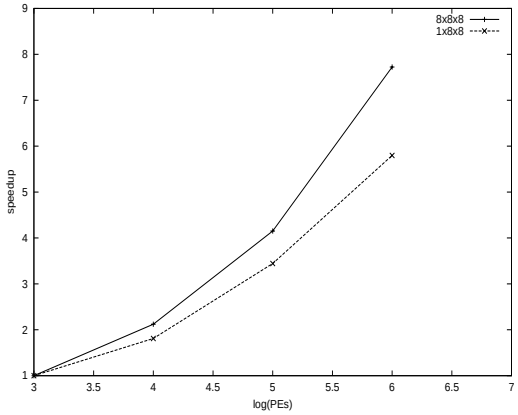

FIG. 3. Speed-up as a function of processor number for two different problem sizes.

V. RESULTS

As a test case we applied our method to compute the friction and diffusion coefficients for a Maxwellian velocity distribution, the results of which are shown in Figs. 4 - 6. The asymptotic fall-off in F_d/v as $1/v^3$ at large v is seen nicely in Fig. 6. An important point that is also clearly demonstrated is the modest number of particles needed per spatial super-cell to reach convergence of the computed quantities.

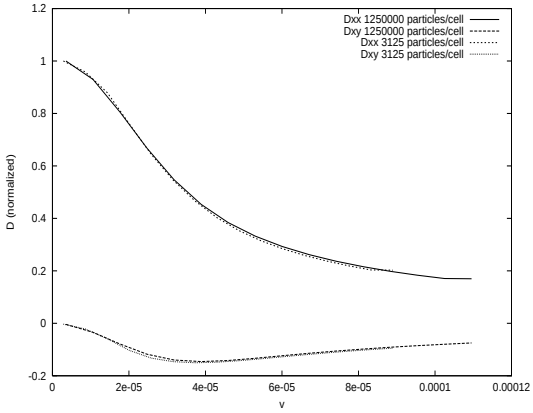

FIG. 4. Diagonal and off-diagonal diffusion coefficients for a Maxwellian distribution as a function of velocity. The expected fall-off in the velocity is clearly seen and excellent results are obtained even for a small number of sampled particles (there is essentially no difference between 3000 and 1.25 million particles).

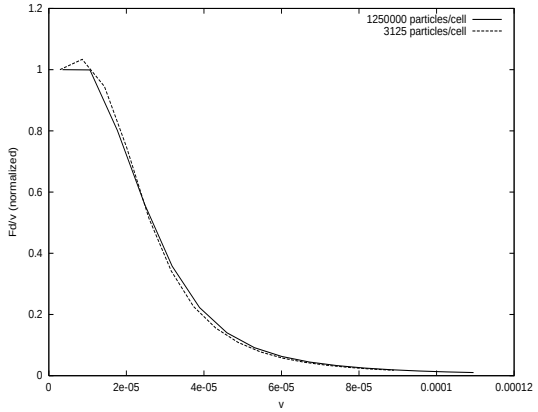

FIG. 5. The friction coefficient divided by velocity as a function of velocity for the same cases as Fig. 4.

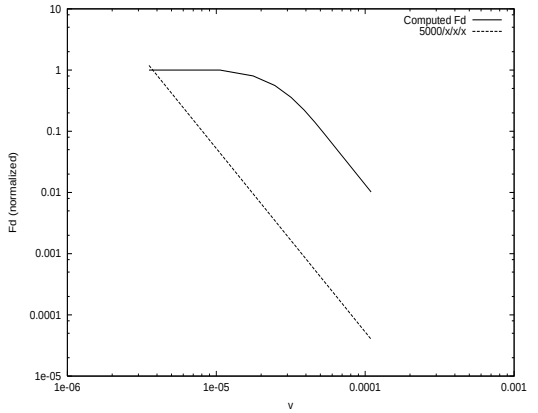

FIG. 6. The friction coefficient divided by velocity as a function of velocity shown on a log-log scale demonstrating the expected $1/v^3$ asymptotic fall-off.

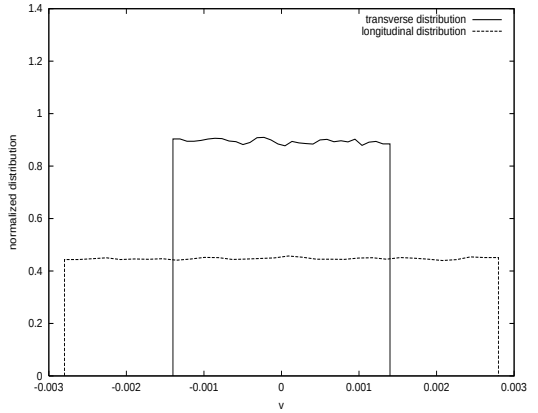

FIG. 7. Transverse and longitudinal initial velocity distributions.

We have also simulated the time evolution of of an initially anisotropic electron beam. The beam has a uniform spatial distribution in a three-dimensional square box with a size of 3.5^3 cm^3. The particle density is $10^{12} cm^{-3}$. The initial velocity distribution is also a step function. Fig.7 gives the initial velocity distribution along the transverse and longitudinal directions. The two distributions have different amplitudes since they have different initial temperature. Fig.8 shows the transverse and longitudinal distributions after 60 steps.

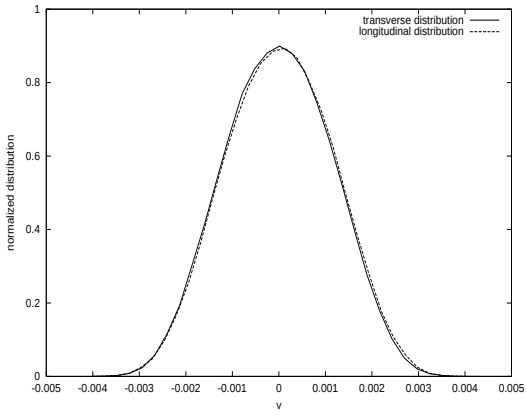

FIG. 8. Transverse and longitudinal initial distributions after 60 steps.

We see that the velocity distributions along both directions approach the thermal Gaussian distributions. Fig.9 shows the time evolution of the temperature in transverse and longitudinal directions as a function of time.

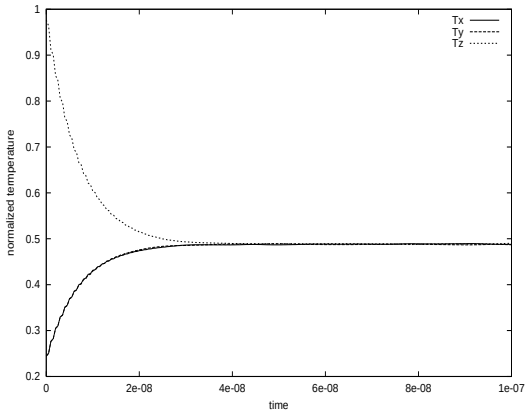

FIG. 9. Transverse and longitudinal temperature as a function of time.

It is seen that the longitudinal temperature gradually decreases, and the transverse temperature gradually increases, to reach an equipartitioned thermal equilibrium due to the intra-beam Coulomb collision.

ACKNOWLEDGMENTS

This work was performed on the SGI/Cray T3E at NERSC, LBNL, and the SGI Origin 2000 at the ACL, LANL. We acknowledge support from the DOE Grand Challenge in Computational Accelerator Physics and the Los Alamos Accelerator Code Group.

---

[1] E.M. Lifshitz and L.P. Pitaevskii, *Physical Kinetics* (Pergamon Press, New York, 1981).
[2] See, e.g., E.C. Shoub, Phys. Fluids **30**, 1340 (1987).
[3] D.R. Nicholson, *Introduction to Plasma Theory* (Wiley, New York, 1983).
[4] See, e.g., J. Qiang, R.D. Ryne, S. Habib, and V. Decyk, SC99 Technical Paper; J. Comp. Phys. (in press).
[5] S. Habib, (unpublished).
[6] M.E. Jones, D.S. Lemons, R.J. Mason, V.A. Thomas, and D. Winske, J. Comp. Phys. **123**, 169 (1996).
[7] W.M. Manheimer, M. Lampe, and G. Joyce, J. Comp. Phys. **138**, 563 (1997).
[8] H. Risken, *The Fokker-Planck Equation: Methods of Solution and Applications* (Springer, New York, 1996).
[9] R.W. Hockney and J.W. Eastwood, *Computer Simulation Using Particles* (Adam Hilger, New York, 1988).
[10] P.C. Liewer and V.K. Decyk, J. Comp. Phys. **85**, 302 (1989).